Phương pháp Lập trình Hướng đối tượng

Tham chiếu

– Từ Giao diện đến Cài đặt –

GV: Lê Xuân Định

“Tham chiếu” trong đời thường

- Chúng ta sử dụng tham chiếu (ref) trong nhiều trường hợp:
  - Đề cập đến (refer) khi nói/viết: “*Mẹ tôi* thương *tôi* lắm!”, “*Lớp trưởng* phân công: bạn *Bằng* lau bảng, bạn *My* mượn micro.”
  - *Số điện thoại*, *nick* trên mạng (chat, forum) được dùng để gián tiếp liên lạc với người khác.
  - *Sóng điện từ / tia hồng ngoại* chiếu từ remote control tới TV
  - *Dây nối* tay bấm tới máy chơi game và tới nhân vật trong game
- Chúng ta tương tác với *đối tượng được tham chiếu* thông qua **giao diện** của nó.
  - “Mẹ tôi”, “lớp trưởng”, số điện thoại, nick → người thật tương ứng
  - Remote control → TV
  - Tay bấm → nhân vật trong game

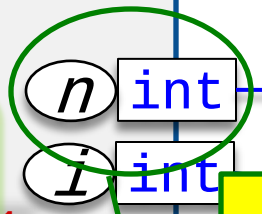
“Tham chiếu” trong đời thường

- Chúng ta tương tác với *đối tượng được tham chiếu* thông qua **giao diện** của nó.
  - “Mẹ tôi”, “lớp trưởng”, số điện thoại, nick → người thật tương ứng
  - Remote control → TV
  - Tay bấm → nhân vật trong game

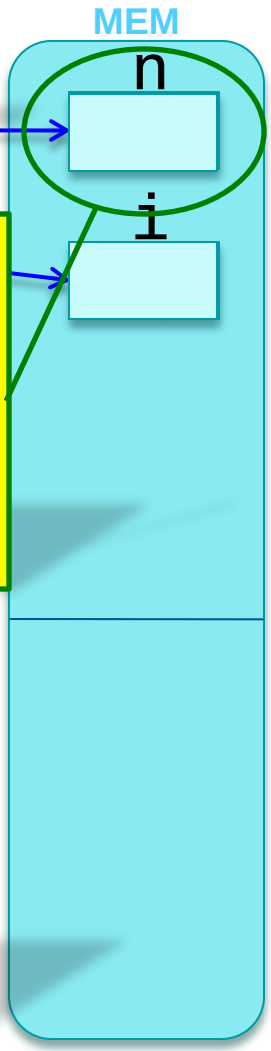
Biến & Tham chiếu

```
void main(){
  int n = 0;
  for(int i=25
    i = (i%2==0)
  }
  int  *p = new int(-1);
  *p = n+1;
  p = &n;
  *p = n+1;
  //char[] a = new char[n];
  char* a = new char[n];
  for(int i=0; i<n; i++){
    a[i] = '#';
  }
}
```

Nhìn từ phía **chương trình**



Tham chiếu

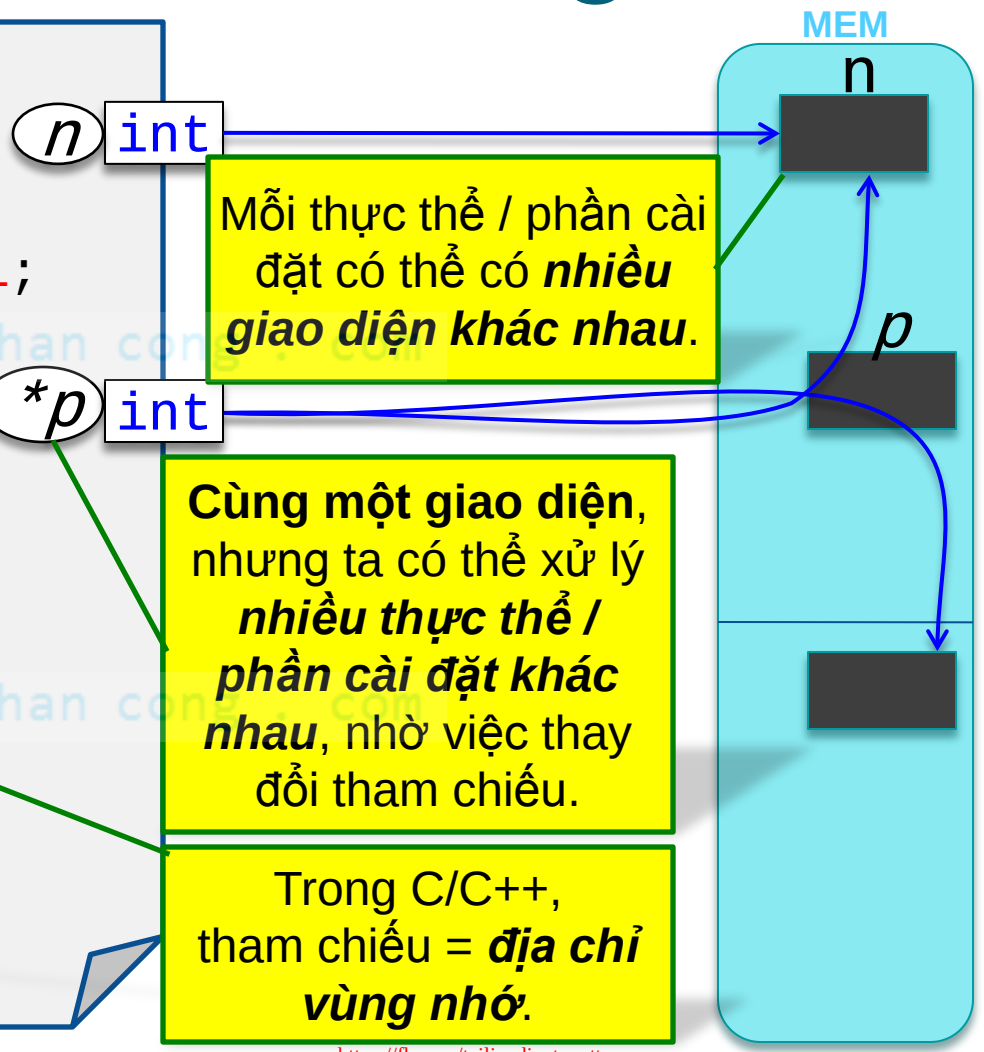


Phần cài đặt := *Vùng nhớ + Giá trị*
là một ***hộp đen***,
nội dung (chính xác)
không hiển thị (tĩnh)
trong chương trình.

Phần giao diện := *Kiểu + Tên biến*
là phần để cho
chương trình *trực tiếp* ***sử dụng***, cung cấp
mọi thông tin (tĩnh)
cần thiết cho ctrình.

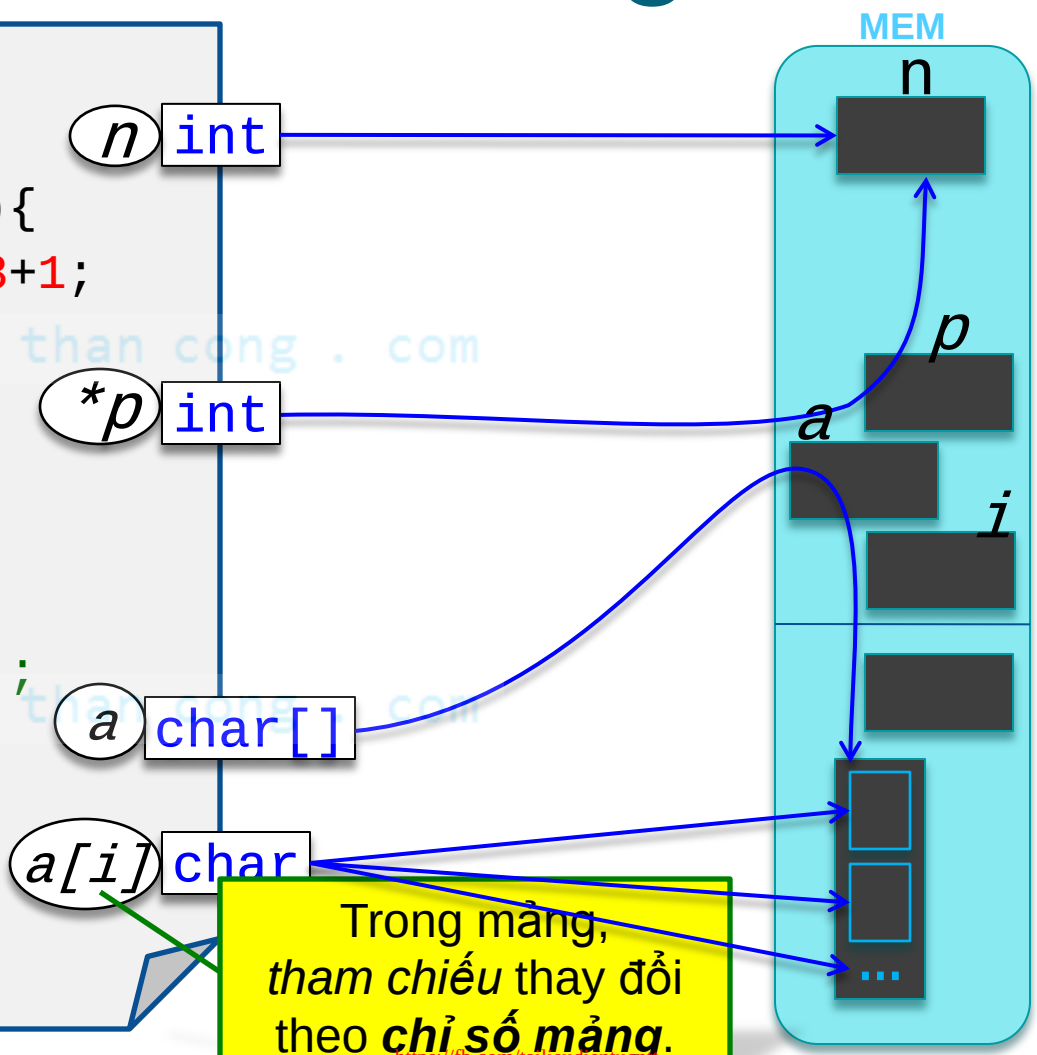
Biến & Tham chiếu động

```
void main(){
  int n = 0;
  for(int i=25; i!=1; n++){
    i = (i%2==0)? i/2: i*3+1;
  }
  int  *p = new int(-1);
  *p = n+1;
  p = &n;
  *p = n+1;
  //char[] a = new char[n];
  char* a = new char[n];
  for(int i=0; i<n; i++){
    a[i] = '#';
  }
}
```



Biến & Tham chiếu động

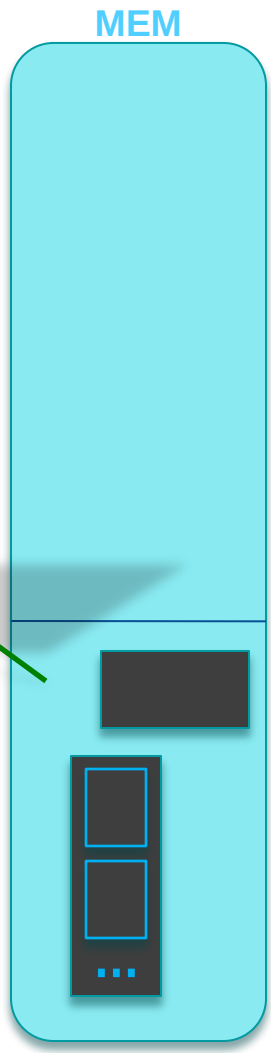
```
void main(){
  int n = 0;
  for(int i=25; i!=1; n++){
    i = (i%2==0)? i/2: i*3+1;
  }
  int  *p = new int(-1);
  *p = n+1;
  p = &n;
  *p = n+1;
  //char[] a = new char[n];
  char* a = new char[n];
  for(int i=0; i<n; i++){
    a[i] = '#';
  }
}
```



Tham chiếu động & Rác

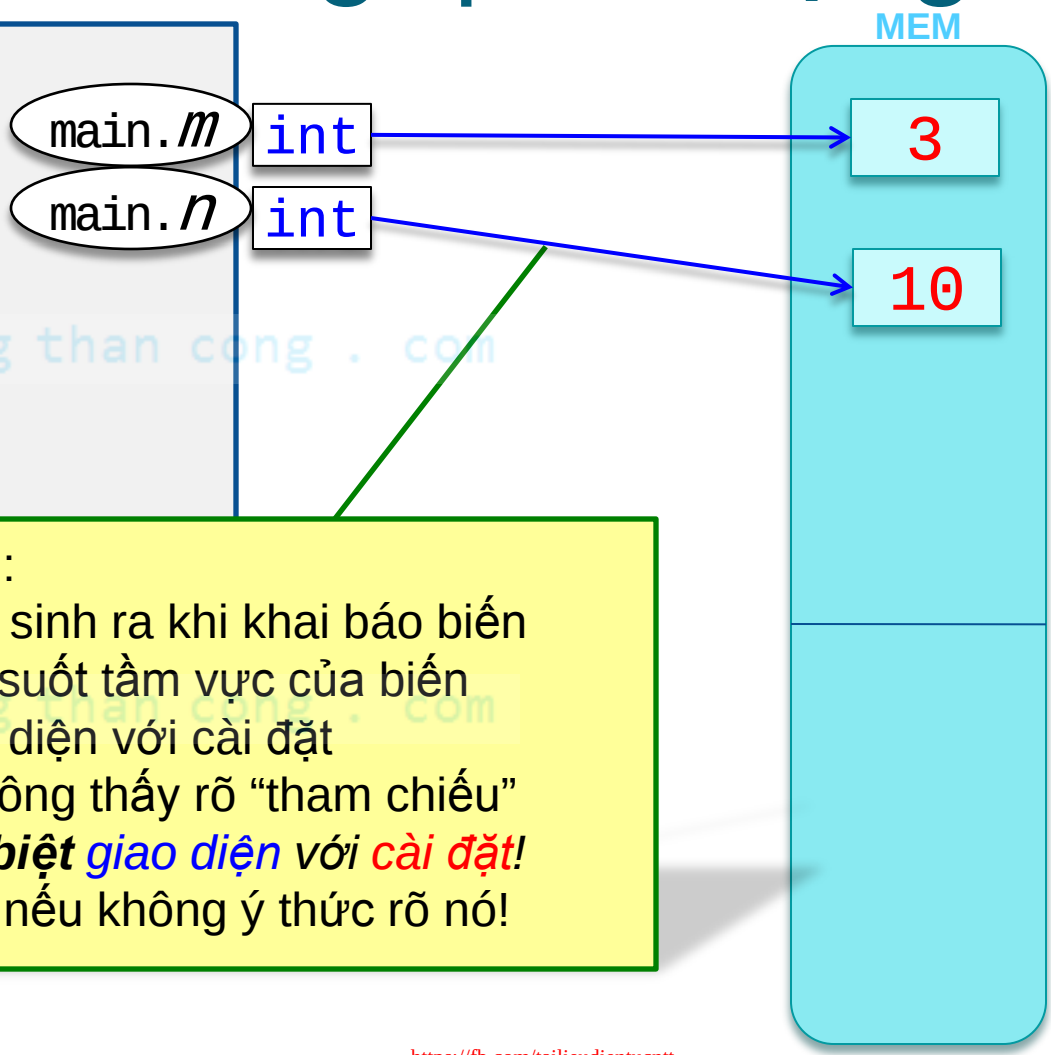
```
void main(){
  int n = 0;
  for(int i=25; i!=1; n++){
    i = (i%2==0)? i/2: i*3+1;
  }
  int  *p = new int(-1);
  *p = n+1;
  p = &n;
  *p = n+1;
  //char[] a = new char[n];
  char* a = new char[n];
  for(int i=0; i<n; i++){
    a[i] = '#';
  }
}
```

Khi dùng biến con trỏ & **cấp phát bộ nhớ động**, rất dễ quên dọn dẹp bộ nhớ (để lại ***rác bộ nhớ***).
Quy tắc
new đi đôi với **delete**.



Tham chiếu tĩnh cũng quan trọng!

```
void main(){
  int n = 10, m = 3;
    cout<<mod(n, m);
    cout<<div(n, m);
}
int mod(int &p, int &q){
    for(; p>q; p -= q);
    return p;
}
```



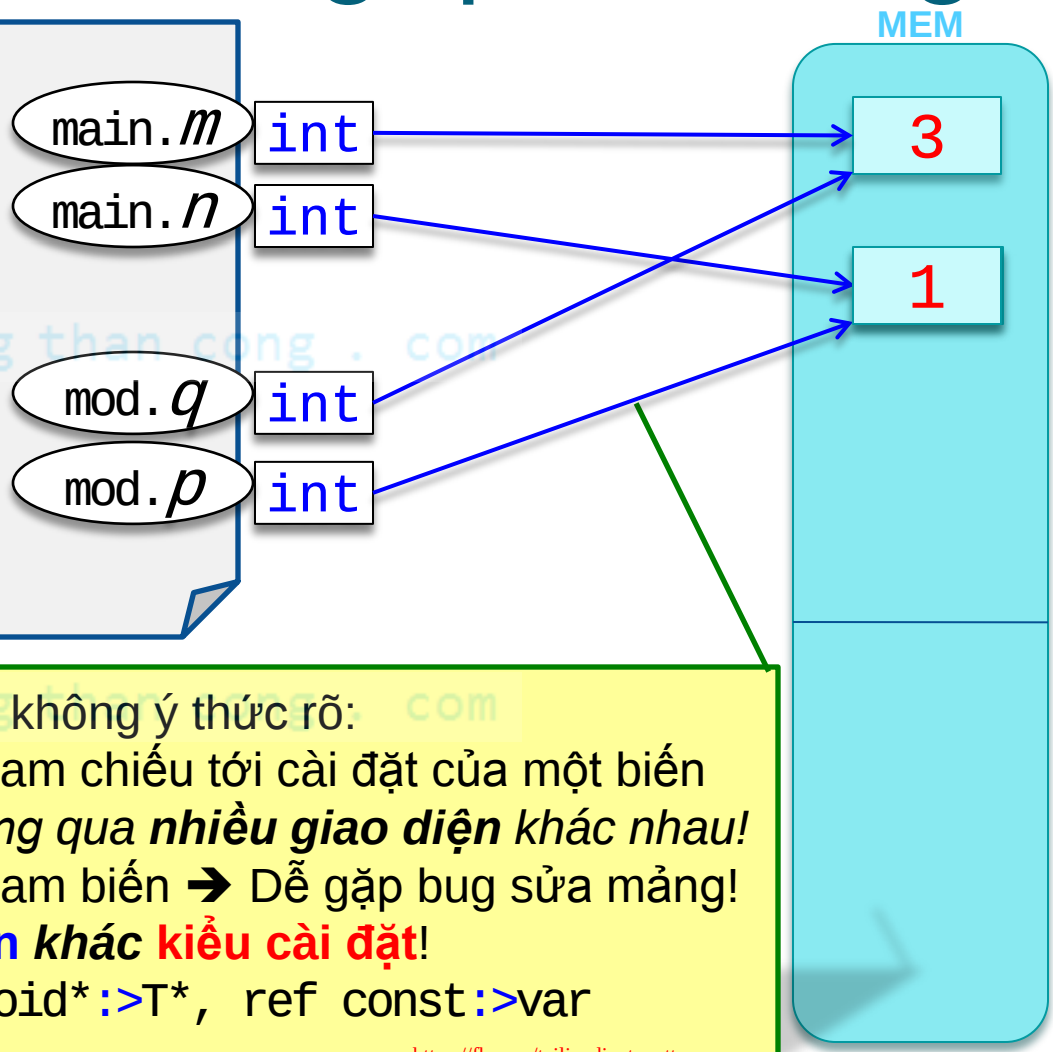
Tham chiếu tĩnh:

- Được tự động sinh ra khi khai báo biến
- Cố định trong suốt tầm vực của biến
- Gắn chặt giao diện với cài đặt
  - ➔ Thường không thấy rõ "tham chiếu"
  - ➔ *Khó **phân biệt** giao diện với cài đặt!*
  - ➔ Nguy hiểm nếu không ý thức rõ nó!

Tham chiếu tĩnh cũng quan trọng!

```
void main(){
  int n = 10, m = 3;
  cout<<mod(n, m);
  cout<<div(n, m);
}
int mod(int &p, int &q){
  for(; p>q; p -= q);
  return p;
}
```

?

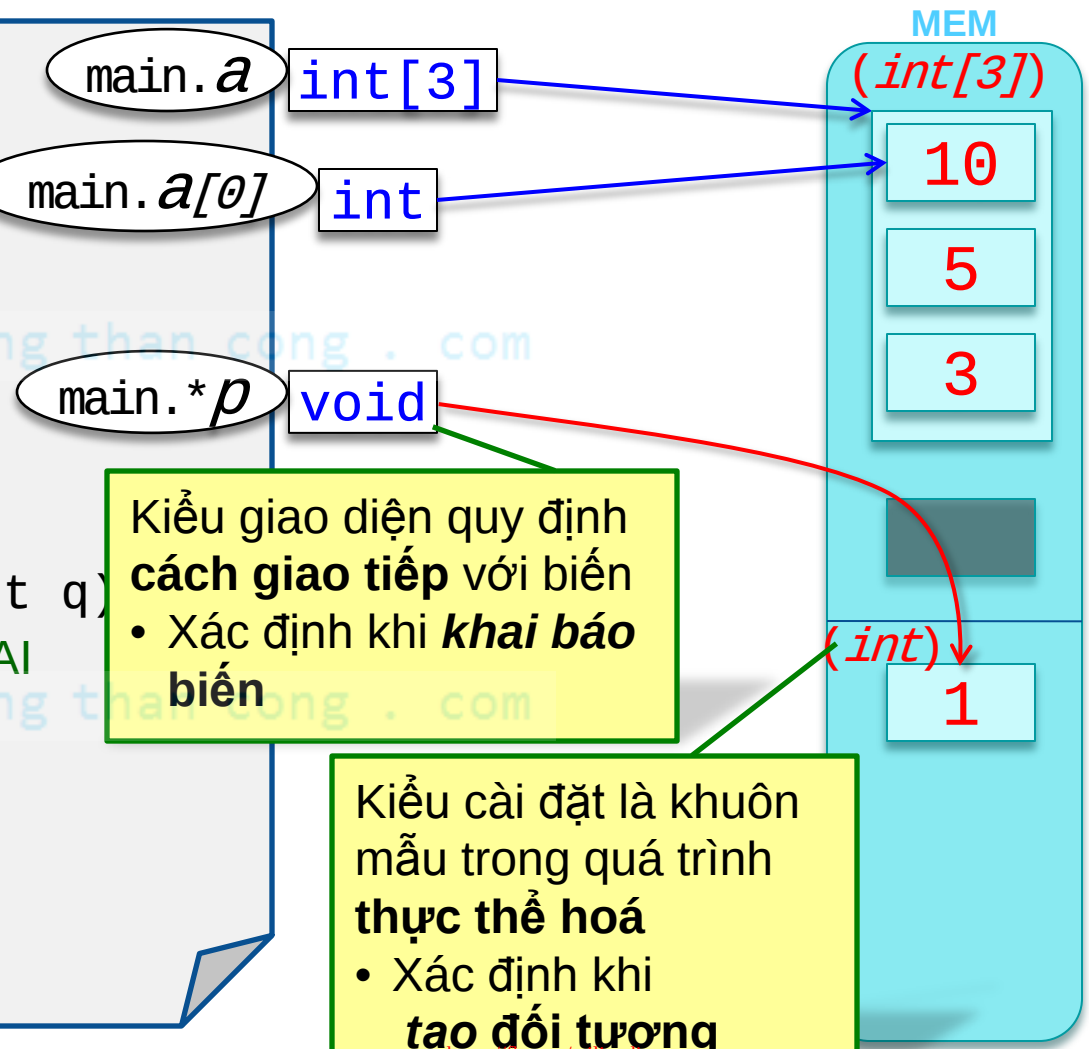


Tham chiếu tĩnh, nguy hiểm khi không ý thức rõ:

- Tham biến là một giao diện tham chiếu tới cài đặt của một biến khác ➔ *Một biến được sử dụng qua **nhiều giao diện** khác nhau!*
- Tham số mảng mặc định là tham biến ➔ Dễ gặp bug sửa mảng!
- Có trường hợp **kiểu giao diện *khác* kiểu cài đặt**!
  - VD: `array[]:>array[N], void*:>T*, ref const:>var`

Kiểu giao diện :> Kiểu cài đặt

```
void main(){
  int a[3] = {10,5,3};
  cout<<mod(a[0],3);
  cout<<sum(a,3);
  cout<<sizeof(a); //12
  void* p = new int(1);
  *p += 2; //SAI
}
int mod(const int &p, int q){
  for(; p>q; p -= q); //SAI
  return p;
}
int sum(int a[], int n){
  cout<<sizeof(a); //4
  ...}
```

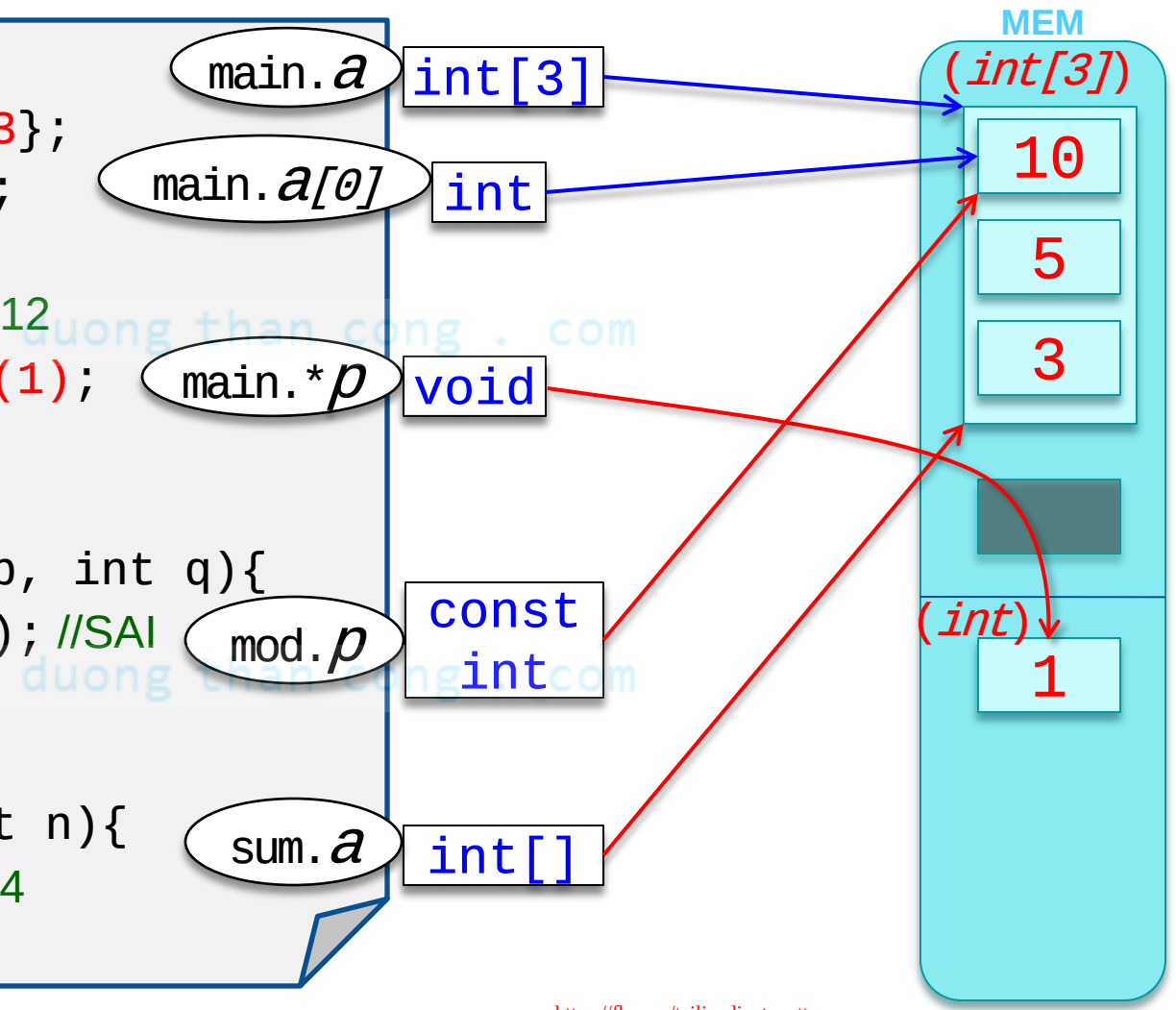


Kiểu giao diện quy định **cách giao tiếp** với biến
- Xác định khi ***khai báo*** **biến**

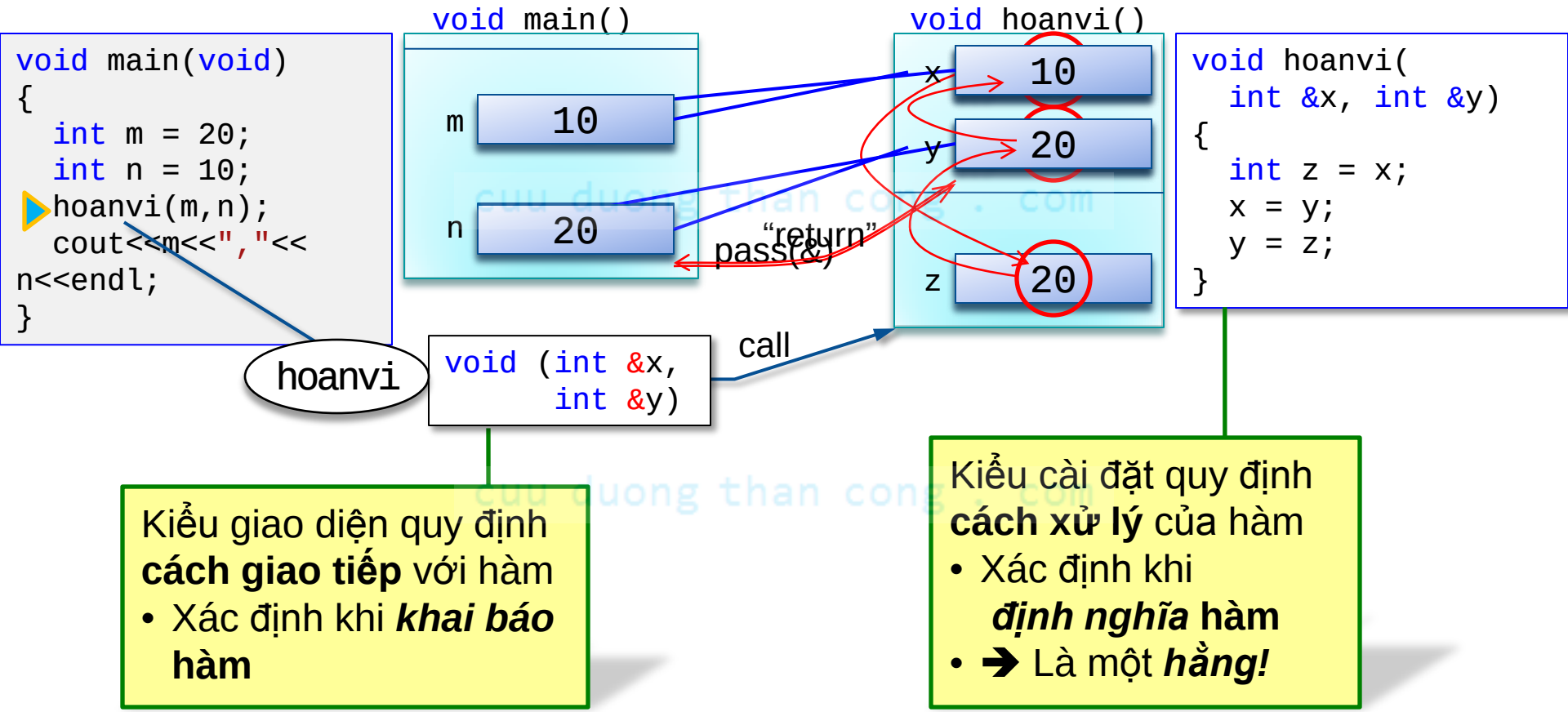
Kiểu cài đặt là khuôn mẫu trong quá trình **thực thể hoá**
- Xác định khi ***tạo đối tượng***

Kiểu giao diện :> Kiểu cài đặt

```
void main(){
  int a[3] = {10,5,3};
  cout<<mod(a[0],3);
  cout<<sum(a,3);
  cout<<sizeof(a); //12
  void* p = new int(1);
  *p += 2; //SAI
}
int mod(const int &p, int q){
  for(; p>q; p -= q); //SAI
  return p;
}
int sum(int a[], int n){
  cout<<sizeof(a); //4
  ...}
```



Hàm & Tham chiếu



Sự Kết buộc Giao diện với Cài đặt

- Sự tạo ra tham chiếu gọi là sự "kết buộc"
  - Là sự gắn kết một phần cài đặt vào cho một giao diện (định danh, id) nào đó.
  - **Kết buộc sớm**: Các id sau được tự động kết buộc bởi trình biên dịch ngay khi khai báo: biến (cục bộ, toàn cục), hàm, tham trị, lớp
    - "Sớm" chứ không phải là "ngay khi viết c.trình"!
    - ***Liên kết lại*** *chương trình* để tái kết buộc.
  - **Kết buộc trễ**: Các id sau được kết buộc bởi chương trình khi chạy chương trình: con trỏ / tham biến tới biến / hàm
    - *Thay đổi tham chiếu* để tái kết buộc.
      - Với con trỏ: *Gán địa chỉ mới*.
      - Với tham biến: *Gọi lại hàm* với đối số là biến/hàm khác.

Tái kết buộc

- Dùng để thay đổi phần cài đặt của một id
  - Thay đổi cài đặt hàm
  - Thay đổi cấu trúc dữ liệu & giải thuật xử lý của đ.tượng
  - Mà không ảnh hưởng đến phần sử dụng id đó!
- Với kết buộc sớm: ***Liên kết lại*** *chương trình*
- Với kết buộc trễ: *Thay đổi tham chiếu*
  - Với hàm: Dùng **tham biến hàm / con trỏ hàm**
  - Với đối tượng: Dùng tham biến / con trỏ đối tượng với ***giao diện đối tượng*** *(lớp thuần trừu tượng trong C++)*

Từ "*Xử lý dữ liệu trong cái hộp*" đến "*Giao tiếp với ĐT qua giao diện*"

- Trước Lập trình Hướng đối tượng
  - Biến là hộp chứa dữ liệu, hàm xử lý dữ liệu chứa trong biến
  - Không cần "tham chiếu", chỉ cần "**con trỏ**"
    - Bản thân con trỏ lại là 1 cái hộp chứa địa chỉ
    - ***Tính toán địa chỉ*** như một số nguyên
  - Không cần phân biệt giao diện với cài đặt biến
    - Khi cần thì ***ép kiểu*** để truy cập / xử lý dữ liệu
- Quan điểm Hướng đối tượng
  - Đối tượng tự quản lý dữ liệu của mình, phân biệt rõ trong/ngoài
  - Thích "**tham chiếu**" (an toàn) hơn "con trỏ" (nguy hiểm)
    - Chỉ quan tâm *cái được tham chiếu tới* (*p), không can thiệp vào cách tham chiếu (bằng địa chỉ trong p, hay truyền tham biến)
  - Phân biệt rõ **giao diện** với **cài đặt** đối tượng
    - Kiểu giao diện được khai báo rõ, không tuỳ tiện ép kiểu!

Giao diện Đối tượng

– Quy định cách giao tiếp với đ.tượng –

"Mỗi hoàn cảnh một giao diện khác nhau"

Tham chiếu đối tượng & Giao diện đối tượng

- Giao diện mặc định của một đối tượng là *tập hợp các thành phần public* của nó
  - Xác định khi khai báo lớp, qua từ khoá `public`
  - Gắn liền với kiểu cài đặt, tức lớp đối tượng
    - Là giao diện được ***kết buộc sớm*** với đối tượng
- Ngoài ra, LTViên có thể định nghĩa thêm các giao diện khác tuỳ theo mục đính sử dụng đối tượng
  - Bao gồm chỉ những *thành phần **thuần giao diện** trừ hàm tạo/huỷ*
  - Được gắn với lớp đối tượng qua khai báo "cài đặt giao diện"
    - Là giao diện được ***kết buộc trễ*** với đối tượng
  - Mỗi giao diện thể hiện **một vai trò** khác nhau của đối tượng
    - Chỉ để giao tiếp với một số đối loại tượng định (đối tác)
    - Chỉ bao gồm những phương thức mà đối tác cần dùng

Tham chiếu đối tượng & Giao diện đối tượng

```
struct IArrInt{ /*interface*/
  virtual int& at(int p)=0;
  virtual int size()=0;
};
class ArrInt :public virtual IArrInt
{public:
  ArrInt(int a[], int n){...}
  ~ArrInt(){...}
  int& at(int p){...}
  int size(){...}
private: ...};

void main(){
  int a[3] = {10,5,3};
  ArrInt ao(a,3);
  IArrInt *ap = &ao;
}
```

Giao diện thuần (C++ gọi "lớp thuần trừu tượng")

- Chỉ có những thành phần ***thuần giao diện***
- Không có hàm tạo/huỷ

Phương thức thuần ảo

- Virtual: "Ảo", tức mang tính giao diện
- "=0": "Thuần", tức hoàn toàn ***không có cài đặt***

MEM

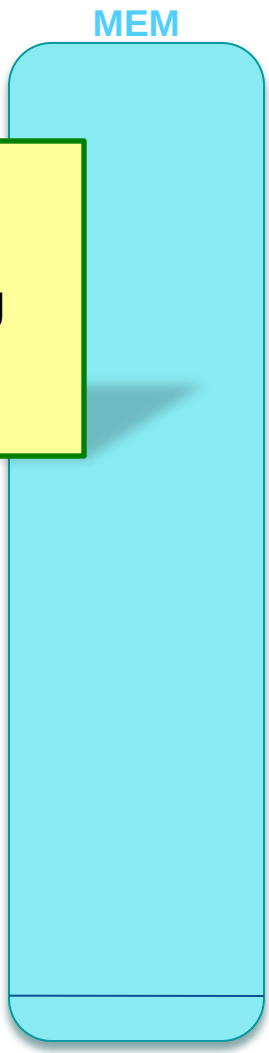
Tham chiếu đối tượng & Giao diện đối tượng

```
struct IArrInt{ /*interface*/
  virtual int& at(int p)=0;
  virtual int size()=0;
};
class ArrInt :public virtual IArrInt
{public:
  ArrInt(int a[], int n){...}
  ~ArrInt(){...}
  int& at(int p){...}
  int size(){...}
private: ...};

void main(){
  int a[3] = {10,5,3};
  ArrInt ao(a,3);
  IArrInt *ap = &ao;
}
```

Lớp `ArrInt` *cài đặt* giao diện `IArrInt`

- Phải có các phương thức khớp với giao diện được cài đặt



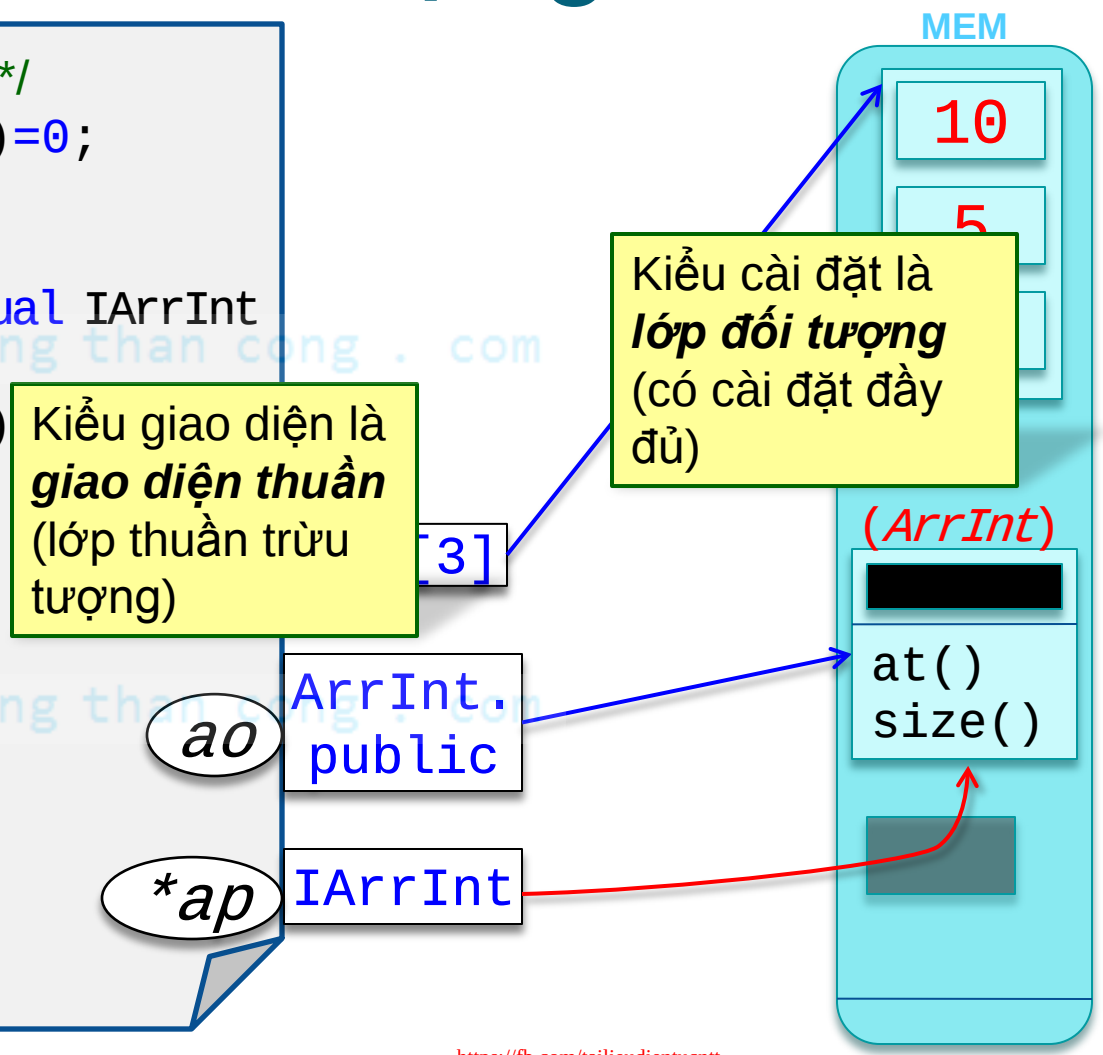
Tham chiếu đối tượng & Giao diện đối tượng

```
struct IArrInt{ /*interface*/
  virtual int& at(int p)=0;
  virtual int size()=0;
};
class ArrInt :public virtual IArrInt
{public:
  ArrInt(int a[], int n)
  ~ArrInt(){...}
  int& at(int p){...}
  int size(){...}
private: ...};

void main(){
  int a[3] = {10,5,3};
  ArrInt ao(a,3);
  IArrInt *ap = &ao;
}
```

Kiểu giao diện là ***giao diện thuần*** (lớp thuần trừu tượng)

Kiểu cài đặt là ***lớp đối tượng*** (có cài đặt đầy đủ)



Hiện tượng Đa hình

Chuối nào cũng là chuối
(cũng ăn được),
nhưng chuối già bự hơn chuối cau
(nhưng chuối cau ăn đã hơn chuối già :D)

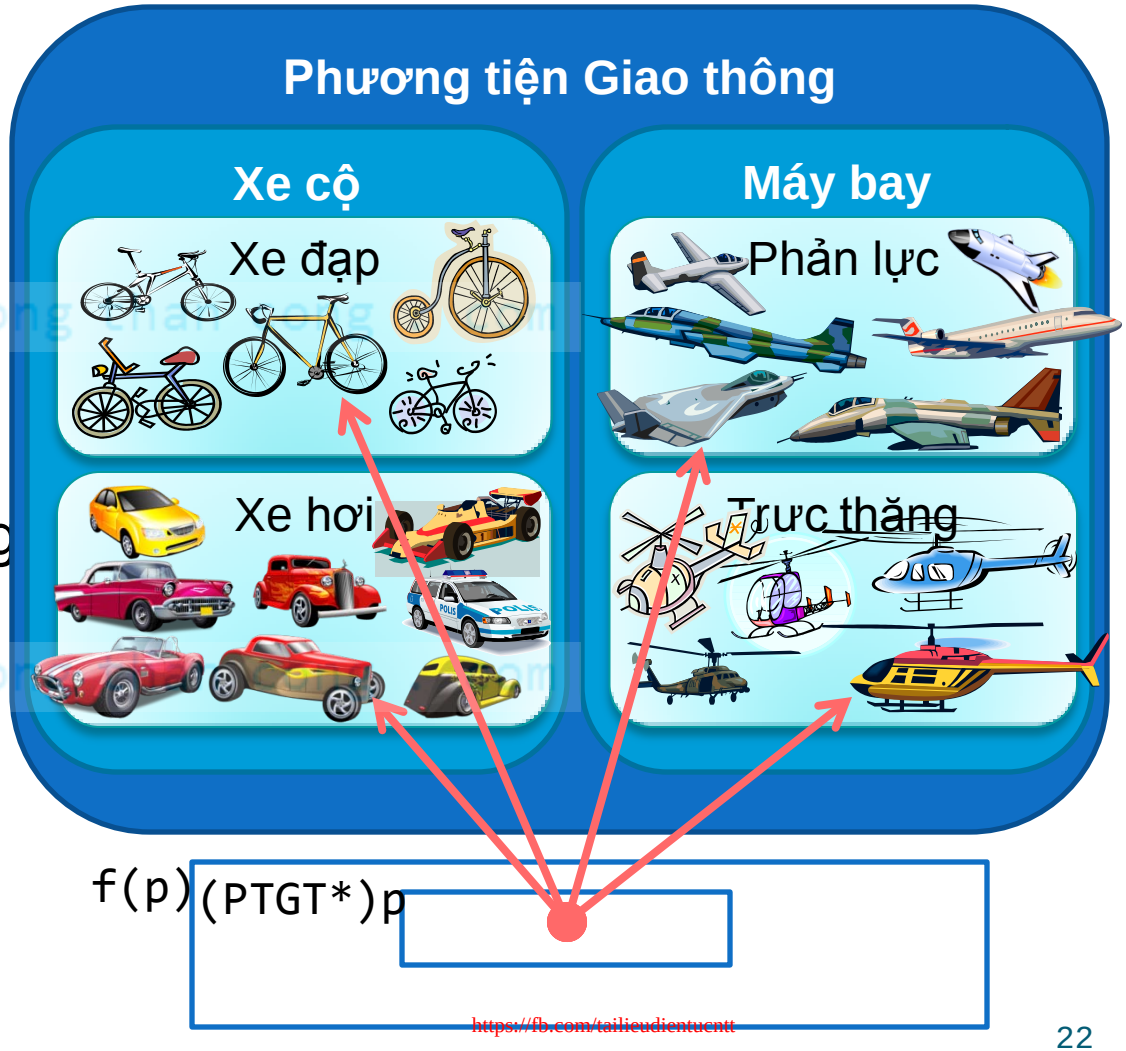
Xe nào cũng là xe
(cũng đi được),
nhưng xe hơi to hơn xe đạp
(nhưng xe đạp đi khoẻ hơn xe hơi :p)

"Chuối nào cũng là chuối!

Cũng có thể ăn được!"

- Nhiều lớp đối tượng có chung một giao diện
  - *Xe đạp*, *xe hơi* đều là *xe*.
  - *Xe cộ* hay *máy bay* đều là *phương tiện giao thông*
    - Cả thảy đều có **giao diện *phương tiện giao thông***.
- 1 biến **tham chiếu** có kiểu là một **giao diện chung** có thể *tham chiếu đến* đ.tượng thuộc ***bất kỳ lớp nào*** *có giao diện đó*.
  - Đối xử với chúng như nhau:

```
void f(PTGT* p) {
  p->diChuyểnĐến("ĐHKHTN");
  ...  }
```



"Chuối già bự hơn chuối cau!

Chuối cau ăn đã hơn chuối già!"

- 1 biến **tham chiếu** có kiểu là một **giao diện chung** có thể *tham chiếu đến* đ.tượng thuộc ***bất kỳ lớp nào*** *có giao diện đó*.
  - Đối xử với chúng như nhau:

```
void f(PTGT* p) {
  p->diChuyểnĐến("ĐHKHTN");
  ...  }
```

- Nhưng cách **phản ứng** (hành động) của đối tượng *thuộc các lớp khác nhau thì* ***khác nhau***.

Bao bọc bằng giao diện ĐT

- **Đối tượng hàm (Functor)**: Khi cần truyền hoặc lưu trữ ***hành động***, ta bọc nó trong một lớp. VD:

GDiện
```
interface ICompare{ bool compare(int i, int j); };
```
Cài Đặt
```
class LessThan :public virtual ICompare
{ bool compare(int i, int j){return i<j;} };
class GreaterThan :public virtual ICompare
{ bool compare(int i, int j){return i>j;} };
```
SDụng
```
interface IOrderedList{ void sort(ICompare& order); ...};
```

- **Đối tượng Nhà máy (Factory)**: Để truyền hoặc lưu trữ *hành động* ***tạo đối tượng*** *của một/nhiều lớp*. VD:

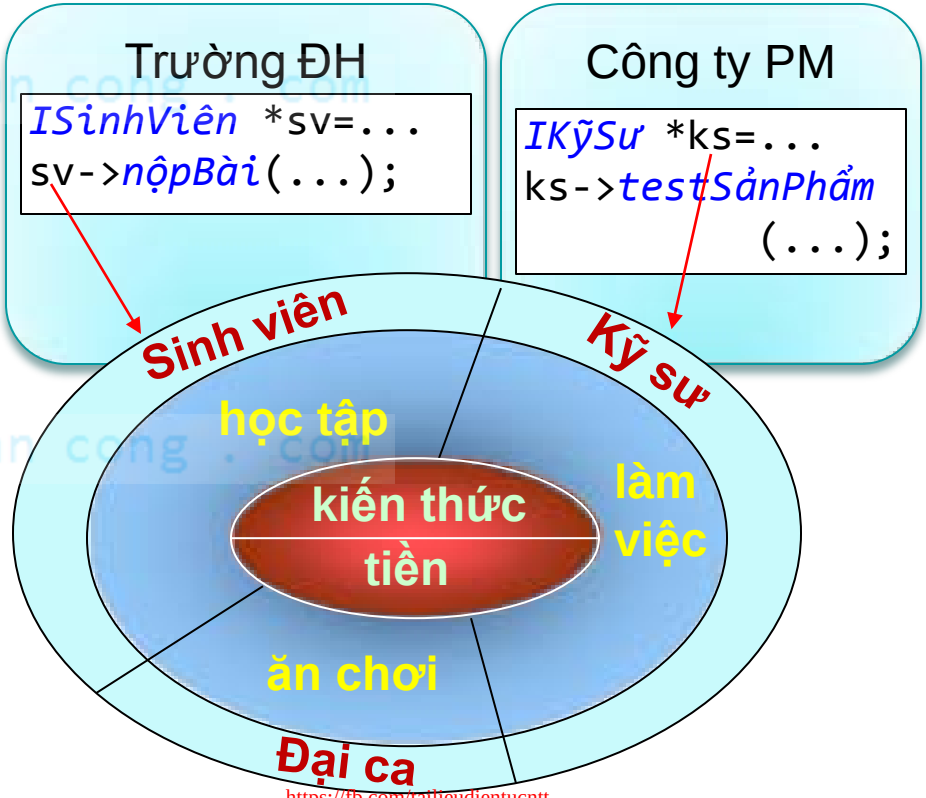
GDiện
```
interface IMyObjFactory{ IMyObj* newInstance(); };
interface IMyObj{ int get(); void set(int a); };
```
Cài Đặt
```
class MyObj :public virtual IMyObj {public: MyObj();...};
class MyObjFactory :public virtual IMyObjFactory
{ IMyObj* newInstance(){return new MyObj();} } fac;
```
SDụng
```
void main(){ IMyObj* mo=fac.newInstance(); mo->set(3); ...};
```

Đa giao diện

- Một đối tượng có thể có nhiều giao diện khác nhau để giao tiếp trong nhiều tình huống khác nhau.

```
class Tui
  :public virtual ISinhViên,
   public virtual IKỹSư,
   private virtual IĐạiCa
{ void nộpBài(...);
  void testSảnPhẩm(...);
  ... };
class TrườngĐH{ ...
  void nhận(ISinhViên &sv); };
class CôngtyPM{ ...
  void nhận(IKỹSư &ks); };
void main(){
  TrườngĐH dh; CôngtyPM cty; ...
  Tui tui; ...

}
```



Bảng đối chiếu thuật ngữ

Tiếng Việt	Tiếng Anh	Chú thích
Tham chiếu	Reference	
Kết buộc, tái kết buộc, kết buộc sớm/trễ	Bind, rebind, early/late binding	
Giao diện (lập trình)	(Programming) interface	
Phần cài đặt	Implementation	
Con trỏ, con trỏ hàm	Pointer, function pointer	
Tham biến	Pass-by-reference param	
Đa hình	Polymorphism	Còn gọi "đa xạ"
Liên kết (chương trình)	(Program) linking	
Lớp trừu tượng	Abstract class	
Định danh	Identifier (id)	Gồm tên biến, tên hàm, v.v.
Ép kiểu	Type cast	